# Zeptoforth on the MAKER PI PICO

Date: 2023-02-11 Author: Christian Hinse

## The purpose of this presentation

The purpose of this presentation is to:

- **install zeptoforth**,
- **introduce the zeptoforth implementation,**
- **demonstrate basic use of zeptoforth.**

For this presentation, I am using:

- a Cytron MAKER PI PICO board,
- a Windows 10 based PC,
- a web browser based terminal emulator and editor provided with zeptoforth.

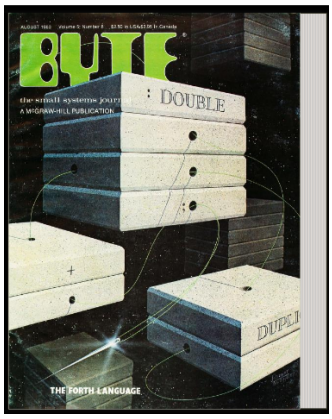
## Who am I?

Before I give you an answer, I will tell you who I am so that my answer can be interpreted in the proper context.

- I am a **college trained electronic technician** who installed, maintained and managed PC based servers and networks in large computer centers.
- I **had home computers** since they appeared in the early 80s: **ZX81** and **Apple II** clone and obviously later an **IBM PC** clone.
- I have **no formal computer science training**.

## My introduction to Forth

I became **addicted to Forth** when I discovered it in the August 1980 Byte magazine issue dedicated to **Forth**.



## My first Forth friends

I like **experimenting with various implementations of Forth** to see the advantages/disadvantages of each implementation. I have been exploring Forth implementation on and off since Forth became available on MCU boards in the early 2000s.

My Forth experimentation so far was done with the following MCU devices.

Single-core MCUs

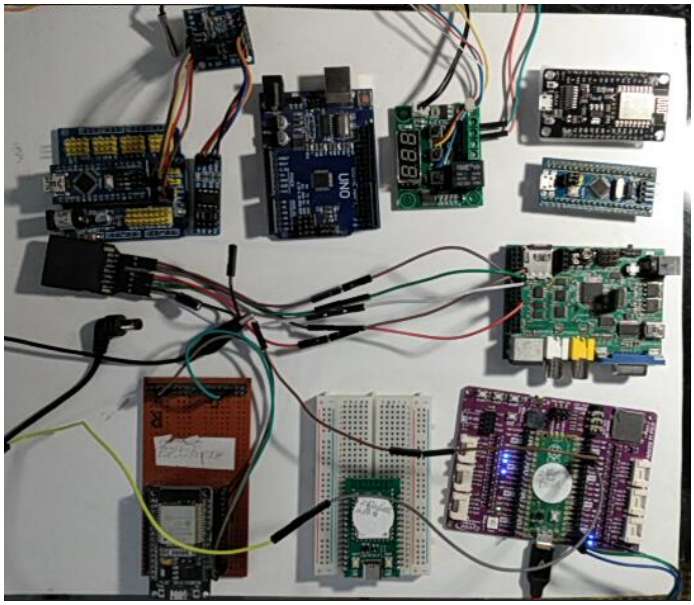
- **Flashforth** and **amforth** on Arduino UNO and NANO boards,
- **STM8 eForth** on a modified W1209 thermostat board,
- **Punyforth** on a Node Mcu ESP8286 based board,
- **Mecrisp-Stellaris** on STM32F103 Bluepill boards,
- **Mecrisp-Stellaris** on the RP2040 PI PICO board.

Multi-core MCUs

- **Tachyon** on the C3 Propeller1 based board,
- **ESP32forth** on ESP32 based boards.
- and now **zeptoforth** on RP2040 based boards.

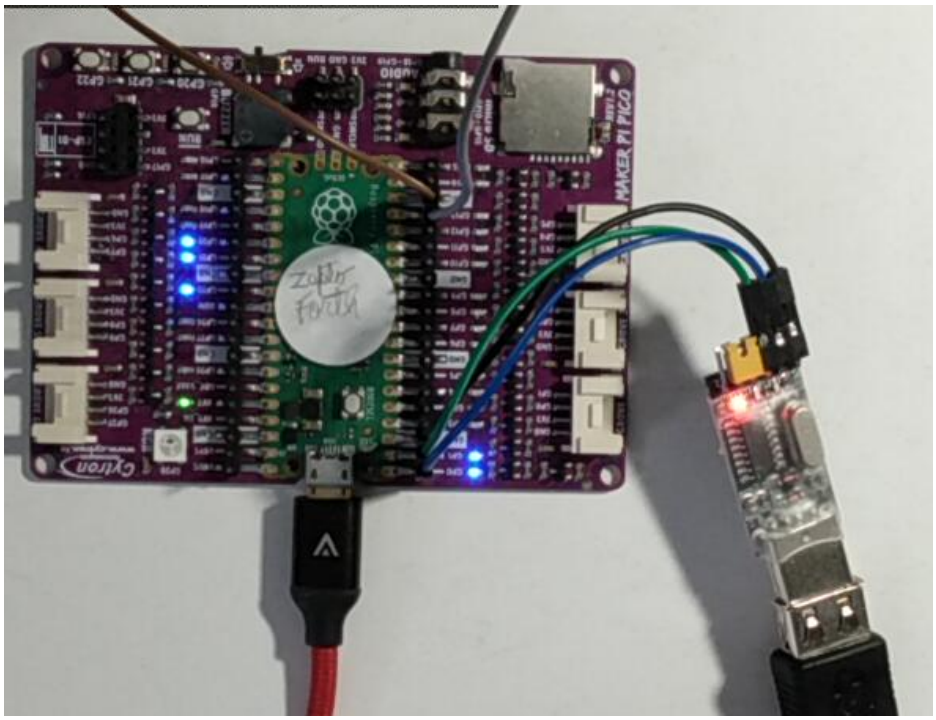
You probably already understood that I am more interested in trying various **Forth implementations** on various **MCU architectures** than **doing** something practical with Forth.

But other members in the Forth2020 group are **good at doing things with Forth.**



## My new Forth friend

Z**eptoforth** on the **Cytron MAKERPI PICO** board is my new Forth friend.




The reasons for choosing **zeptoforth** for my current Forth exploration are purely personal.

- It appears to be so much **more complete** than other Forth implementations I experimented with.
- It includes many of the **current computer science concepts** that I always wanted to understand.
- It is **available on a low-cost MCU** like the RP2040 based PI PICO.
- It has **programmable IO processors**.
- Zeptoforth on the RP2040 allows **multi-processing** with the **2 cores**.
- The **installation is simple** on the PI PICO also named RP2040.
- Zeptoforth **documentation is excellent**.
- It is **still evolving** at a very fast pace and is **well supported** by **Travis Bemann.**
- It provides a simple minimal terminal/editor **web browser based development environment.**
- Zeptoforth is **mature and stable**.

# Zeptoforth overview

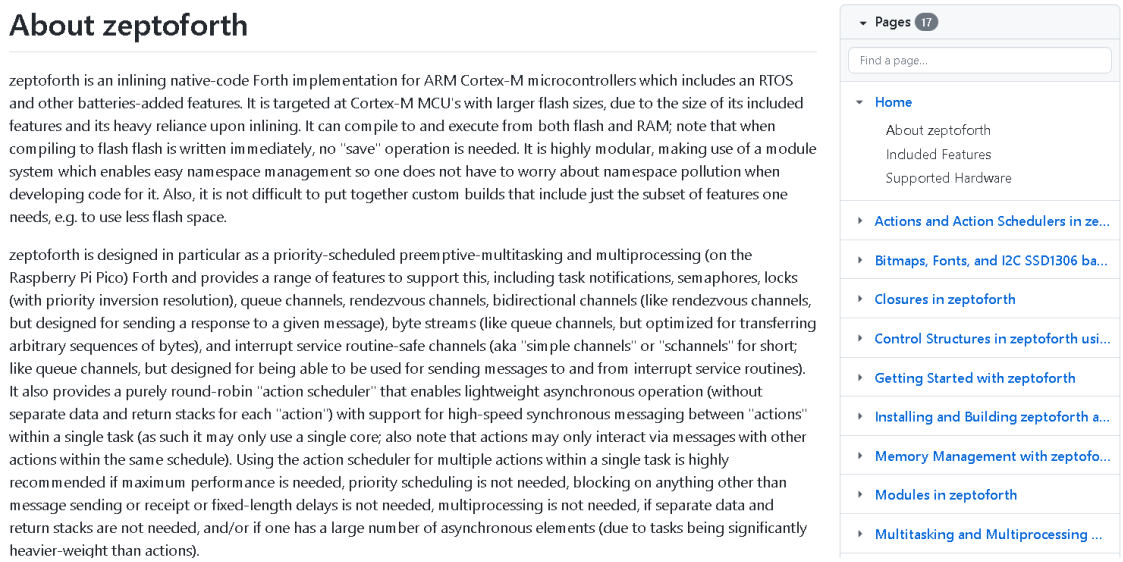
Zeptoforth was created and is still supported and developed by **Travis Bemann**.

The information below was extracted from his main zeptoforth web site.

## Main zeptoforth web site

This presentation is just a summary. For more details refer to the main web site. This web site is the **reference for all zeptoforth documentation** which is very well done.

A glance at the main zeptoforth web site



### About zeptoforth

zeptoforth is an inlining native-code Forth implementation for ARM Cortex-M microcontrollers which includes an RTOS and other batteries-added features. It is targeted at Cortex-M MCU's with larger flash sizes, due to the size of its included features and its heavy reliance upon inlining. It can compile to and execute from both flash and RAM; note that when compiling to flash flash is written immediately, no "save" operation is needed. It is highly modular, making use of a module system which enables easy namespace management so one does not have to worry about namespace pollution when developing code for it. Also, it is not difficult to put together custom builds that include just the subset of features one needs, e.g. to use less flash space.

zeptoforth is designed in particular as a priority-scheduled preemptive-multitasking and multiprocessing (on the Raspberry Pi Pico) Forth and provides a range of features to support this, including task notifications, semaphores, locks (with priority inversion resolution), queue channels, rendezvous channels, bidirectional channels (like rendezvous channels, but designed for sending a response to a given message), byte streams (like queue channels, but optimized for transferring arbitrary sequences of bytes), and interrupt service routine-safe channels (aka "simple channels" or "schannels" for short; like queue channels, but designed for being able to be used for sending messages to and from interrupt service routines). It also provides a purely round-robin "action scheduler" that enables lightweight asynchronous operation (without separate data and return stacks for each "action") with support for high-speed synchronous messaging between "actions" within a single task (as such it may only use a single core; also note that actions may only interact via messages with other actions within the same schedule). Using the action scheduler for multiple actions within a single task is highly recommended if maximum performance is needed, priority scheduling is not needed, blocking on anything other than message sending or receipt or fixed-length delays is not needed, multiprocessing is not needed, if separate data and return stacks are not needed, and/or if one has a large number of asynchronous elements (due to tasks being significantly heavier-weight than actions).



### Supported Hardware

Zeptoforth is a Cortex-M Forth. It currently supports the following boards:

- Raspberry **Pi Pico and other RP2040-based boards** STM32F407 DISCOVERY
- STM32L476 DISCOVERY
- STM32F746 DISCOVERY
- STM32F411 "Black Pill"

Currently, I am only interested in the **RP2040** based implementation.

**Included Features**

In the feature list below, it is obvious that zeptoforth was created by someone who is **computer science oriented**.

Personally, I think **zeptoforth** is an **upgraded Forth implementation** addressing the **current CS concepts** required for the **current multi-processor MCU** devices.

**I will be honest, I don't yet understand most of the CS features listed below.**
But this implementation of Forth will probably allow me to learn a lot about **CS and new MCU architectures.**

MCU concepts

Popular MCU concepts are all there.

- A priority-scheduled preemptive multitasker.
- Double cell and fixed point numeric support.
- Values and lexically scoped local variables.
- A disassembler.
- SysTick support.
- Interrupt-driven serial IO.
- **GPIO** support (including EXTI support on STM32 microcontrollers).
- General UART support (in addition to serial console support).
- One-shot ADC support.
- **SPI** (both master and slave) support.
- I2C (both master and slave) support (on the RP2040).
- PWM support (on the RP2040).
- Hardware timer support (on the RP2040).
- **RTC** support (on the RP2040).
- LED drivers.
- **SDHC/SDXC** card support using SPI.
- **FAT32 support** on SDHC/SDXC cards.
- Support for code loading from files in FAT32 file systems.
- User-level FAT32 tools.
- **Quad SPI flash storage** support (on the STM32F746 DISCOVERY board and the RP2040).
- Random number generator support (except on the STM32F411 "Black Pill" and STM32F411 Nucleo 64 boards).
- Pseudorandom number generator support (using the TinyMT32 PRNG)
- **Programmable input/output support** (on the RP2040).

Computer science concepts

Most of the computer science concepts that I never understood are in there.

- **Semaphores**.
- **Locks**, with priority inversion handling.
- Message-oriented **queue** channels.
- Message-oriented rendezvous channels.
- Message-oriented bidirectional synchronous reply **channels**.
- Byte-oriented **streams**.
- Interrupt service handler-safe channels.
- **Task notifications**.
- **Action scheduler** support.
- **Multicore support** (on the RP2040).
- **Lambda** expressions.
- **Closures**.
- **Object orientation**.
- User-reconfigurable processor **exception vector** support.
- Rebooting via control-C on the console.
- **Maps**, including counted string and integer-keyed maps.
- **Heap** allocators.
- **Memory pool** allocators.
- **Task pool** allocators.
- **Action pool** allocators.
- Temporary storage allocators (used to provide immediate string constants).
- Best-effort fault recovery.

Development tools

It even has some nice development tools.

- A line editor.
- A block editor (on the STM32F746 DISCOVERY board and the RP2040).
- **A web browser based terminal emulator/editor**.
- Optional swdcom support.
- Optional task monitor.

### Material used in the presentation

For this presentation, I will be using:

- a Cytron MAKER PI PICO board,
- a Windows 10 based PC,
- a **web browser based terminal emulator and editor** provided with zeptoforth:

  zeptocom.js

To understand any MCU, the Technical reference manual is a must to answer questions when required. You can download the data sheet PDF file at the link below.

RP2040 data sheet

## Zeptoforth installation

The zeptoforth installation process with a pre-compiled binary **uf2** file is **very simple** on an RP2040 PI PICO.

- **Get the ZIP** file from the following **github** site: https://github.com/tabemann/zeptoforth/releases/tag/v0.60.0 . Click on **zeptoforth-0.60.0.tar.gz** to download the compressed file.
- Open the downloaded file with your favorite compress/decompress software and **browse to the folder containing the pre-compiled uf2 binary files**: E:\Christian\Downloads\zeptoforth-0.60.0.tar.gz\zeptoforth-0.60.0.tar\zeptoforth-0.60.0\bin\0.60.0\rp2040\
  I use 7-Zip. In the above folder path replace E:\Christian with your own disk drive letter and your name.
- In the just opened folder, **decompress the file zeptoforth_full-0.60.0.uf2** into your Downloads folder. In 7-Zip you select the desired file and click on Extract.
- **Open your file browser and locate the zeptoforth_full-0.60.0.uf2** in your Downloads folder.
- **Enter the RP2040 bootload mode** by powering-up the RP2040 board while depressing the BOOTSEL button and releasing it when you PC indicates that the **RPI-RP2 USB** drive has been created.
- **Open the RPI-RP2 USB drive with your file browser** and place the browser window beside your previously opened Downloads browser window.
- **Drag and drop the zeptoforth_full-0.60.0.uf2 file into your RPI-RP2 USB drive.** The **zeptoforth_full-0.60.0.uf2** file will be uploaded into the RP2040 and the RPI-RP2 USB drive will disappear.

Your RP2040 is now running **zeptoforth**. You now have to start a terminal emulator to access your RP2040 serial port and use it.

## Accessing the RP2040 serial port

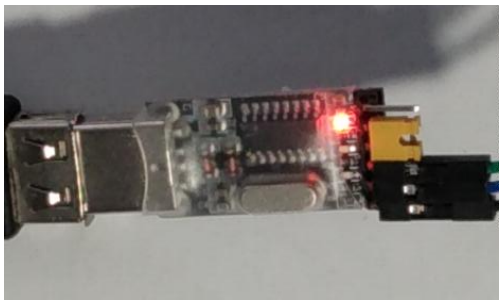
Zeptoforth also provides a **web browser based terminal emulator and editor** to access the RP2040 serial port and start programming with zeptoforth. But the RP2040 has a weakness, it does not provide a USB serial port like most other MCU devices.

**<u>Connecting the USB TO TTL adapter</u>**

You will have to use a **USB TO TTL serial adapter** to connect to the RP2040 serial port. The RP2040 hardware includes a USB interface but it is only used for the loading of the pre-compiled binary uf2 file. The required software to use it as a serial port is still to be developed for Forth.

The RP2040 MCU operates at 3.3V and your USB to serial adapter must be configured to operate with RX/TX signals at 3.3V. On my YP-02 USB TO TTL adapter, it is done by placing a jumper between VCC and 3V3 on the adapter pins. Remember that the adapter RX goes to the RP2040 TX and adapter TX to RP2040 RX.

Here is a picture of the YP-02 USB TO TTL adapter used to connect to the Cytron MAKER PI PICO serial port.



Connect the USB TO TTL adapter to the RP2040 as indicated below.

| ***USB TO TTL adapter*** | ***RP2040*** |
|---|---|
| TX | RX on GP0 pin 1 |
| RX | TX on GP1 pin 2 |
| GND | GND on pin 3 |

We are now ready to start the **web browser based terminal emulator and editor** to configure it and start programming the RP2040 with **zeptoforth**.

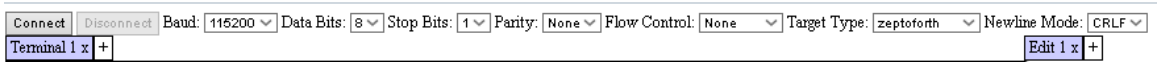
## Configuring the web browser based terminal emulator and editor

In your favorite web browser, start the **zeptoforth web browser based terminal emulator and editor** by clicking on the following internet link:

zeptocom.js

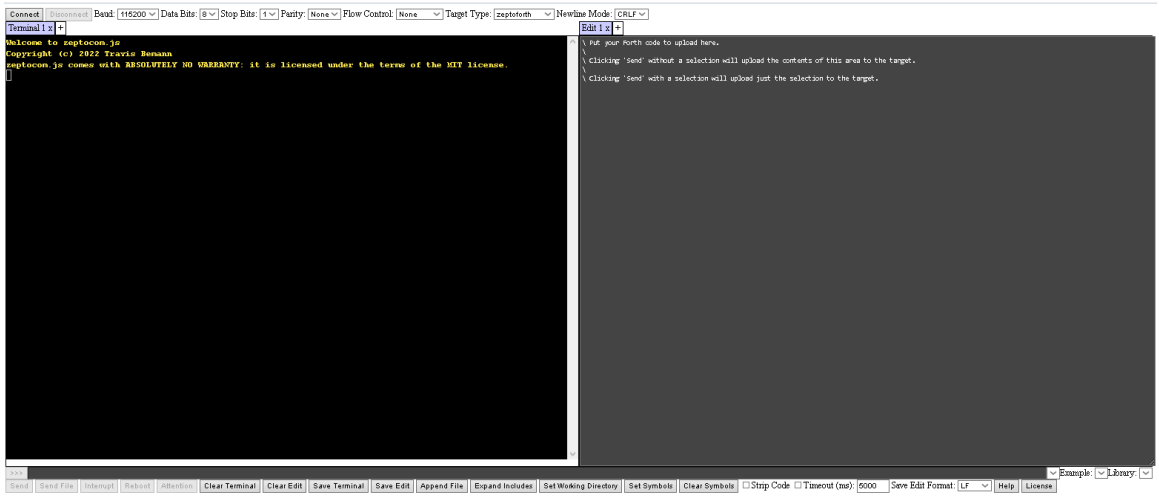
You should now see the following web page. Use the default settings shown at the top of this screen.

***Enlarged settings***




## Using the web browser based terminal emulator and editor

***Main screens***




## In a summary

- The **Edit pane** is used to enter the Forth code. The **Send button** sends the Edit screen content to the Terminal screen serial port.
- The **Terminal pane** shows what is happening on the RP2040 serial port.
- The **data entry field** between the **Terminal pane** and the **Send, Send File** functions is used to enter a **short command**.
- The **various buttons** at the bottom of the screens allow loading and saving files between your PC disk and the Terminal and Editor screens.
- The **Help button** shows the function of various buttons in the Terminal pane. I will let you explore the use of theses buttons on your own.

## Using zeptoforth

Zeptoforth provides too much functionality to cover it in detail here.

We will take a quick look at what it can do on the main zeptoforth web site.

## Main zeptoforth web site

You will have to do like me and explore it on your own in the following weeks.

## zeptocom.js

**Some very basic examples**

- enter **short comma**nds from the **data entry field**,

```
- 1 2 3 .s
- + .s
- + .s
```

- enter **programs** from the **Edit pane**,

```
: add-1
  1 2 3 .s
  + .s
  + .s
  . .s
;

: add-2
  1 2 3 .s cr
  + .s  cr
  + .s cr
  . .s cr
;
```

- **load programs** directly from your PC disk drive,
  - RP2040-Load-Blinky-CH1.txt

  Here is the content of this file. Create a file with this content and click on the **Send File button** to select it in the file browser and load it in the RP2040 MCU. If you prefer, you could also **cut and paste** the code in the **Editor pane** and click the **Send button**.

```
\ RP2040 onboard green LED control demo.
led import
: blinky ( -- )
  begin key? not while green toggle-led 500 ms repeat
  key drop
;
blinky
```

  PWM PIO LED fade demo

- **load a program** that controls a LED **with a PIO controller.**
  This example fades the pin 13 LED on and off gradually.
  - zeptoMAKER-piotest-CH1.txt

  Here is the content of this file. Create a file with this content and click on the **Send File button** to select it in the file browser and load it in the RP2040 MCU. If you prefer, you could also **cut and paste** the code in the **Editor pane** and click the **Send button**.

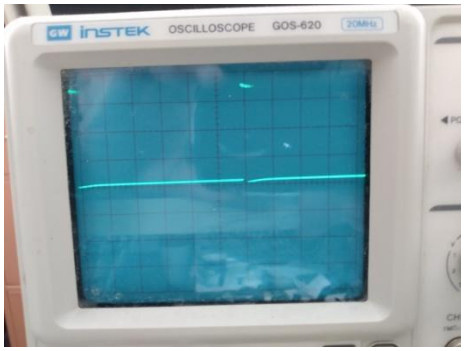
```
\ Copyright (c) 2021-2023 Travis Bemann
\
\ Permission is hereby granted, free of charge, to any person obtaining a copy
\ of this software and associated documentation files (the "Software"), to deal
\ in the Software without restriction, including without limitation the rights
\ to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
\ copies of the Software, and to permit persons to whom the Software is
\ furnished to do so, subject to the following conditions:
\
\ The above copyright notice and this permission notice shall be included in
\ all copies or substantial portions of the Software.
\
\ THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
\ IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
\ FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
\ AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
\ LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
\ OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
\ SOFTWARE.

continue-module forth

  interrupt import
  pio import
  task import
  systick import

  \ The initial setup
  create pio-init
  1 SET_PINDIRS set,
  0 SET_PINS set,

  \ The PIO code
  create pio-code
  PULL_BLOCK PULL_NOT_EMPTY pull,
  32 OUT_X out,
  1 SET_PINS set,
  3 COND_X1- jmp,
  PULL_BLOCK PULL_NOT_EMPTY pull,
  32 OUT_X out,
  0 SET_PINS set,
  7 COND_X1- jmp,

  \ The blinker state
  variable blinker-state

  \ The blinker maximum input shade
  variable blinker-max-input-shade

  \ The blinker maximum shade
  variable blinker-max-shade

  \ The blinker shading
  variable blinker-shade

  \ The blinker shade step delay in 100 us increments
  variable blinker-step-delay

  \ The blinker multiplier
  2variable blinker-multiply
```

```
\ The blinker pre-multiplier
2variable blinker-premultiply

\ The blinker shade conversion routine
: convert-shade ( i -- shade )
  0 swap blinker-max-input-shade @ 0 swap f/ pi f* cos dnegate 1,0 d+ 2,0 f/
  blinker-premultiply 2@ f* expm1 blinker-premultiply 2@ expm1 f/
  blinker-multiply 2@ f* nip
;

\ PIO interrupt handler
: handle-pio ( -- )
  blinker-state @ not if
    blinker-shade @ 0 PIO0 sm-txf!
  else
    blinker-max-shade @ blinker-shade @ - 0 PIO0 sm-txf!
  then
  blinker-state @ not blinker-state !
  PIO0_IRQ0 NVIC_ICPR_CLRPEND!
;

\ Set the shading
: blinker-shade! ( i -- )
  blinker-max-input-shade @ convert-shade blinker-max-shade !
  convert-shade blinker-shade !
;

\ The blinker shading task loop
: blinker-shade-loop ( -- )
  begin
    blinker-max-input-shade @ 0 ?do
     i blinker-shade!
     systick-counter blinker-step-delay @ current-task delay
    loop
    0 blinker-max-input-shade @ ?do
     i blinker-shade!
     systick-counter blinker-step-delay @ current-task delay
    -1 +loop
  again
;

\ Init blinker
: init-blinker ( -- )
  true blinker-state !
```

```
    125 blinker-max-input-shade !
    500,0 blinker-multiply 2!
    5,0 blinker-premultiply 2!
    25 blinker-step-delay !
    0 blinker-shade!
    %0001 PIO0 sm-disable
    %0001 PIO0 sm-restart
    0 758 0 PIO0 sm-clkdiv!
\ For PI PICO board use the following line.
\ 25 1 0 PIO0 sm-set-pins!
\
\ For a MAKER PI PICO use the following line
\ to allow showing the PWM signal on an oscilloscope.
\ GP 13 is easier to connect to and has a LED on the
\ MAKER PI PICO. The WIO RP2040 also uses GP 13 for its onboard LED.
    13 1 0 PIO0 sm-set-pins!
    0 7 0 PIO0 sm-wrap!
    on 0 PIO0 sm-out-sticky!
    pio-init 2 0 PIO0 sm-instr!
    pio-code 8 PIO0 pio-instr-mem!
    0 0 PIO0 sm-addr!
    blinker-shade @ 0 PIO0 sm-txf!
    ['] handle-pio PIO0_IRQ0 16 + vector!
    0 INT_SM_TXNFULL IRQ0 PIO0 pio-interrupt-enable
    PIO0_IRQ0 NVIC_ISER_SETENA!
    %0001 PIO0 sm-enable
    0 ['] blinker-shade-loop 420 128 512 spawn run
  ;

end-module

init-blinker
```

PWM PIO signal on the oscilloscope




- show that the zeptoforth web browser based terminal emulator and editor can also be used to **control other Forth implementations.** Experiment on your own if you have the following Forth implementations.
  - ESP32forth,
  - Mecrisp,
  - STM8 eforth,
  - Flashforth.

## Other versions of zeptoforth

The **zeptoforth** version that I presented is the **full version**.

**Lighter versions of zeptforth** are also available as pre-compiled binaries in the same folder as the full version. They are probably sufficient in less complex applications.

I still have to explore the level of functionality available in the various available uf2 files.

New MCU boards are being added to the supported list. **Travis Bemann** recently ported **zeptoforth** to the **XIAO RP2040**.

I also proved that it also runs on the **WIO RP2040 dev board.**

**Zeptoforth AT commands words** might soon allow the WIO RP2040 dev board to control the integrated Wi-Fi ESP8285 MCU for Wi-Fi access.

**Thanks Travis!**

## Forth is adapting to the current MCU and Computer Science technology!